Configuration Space

Cspace Intro

SECTION 2.1

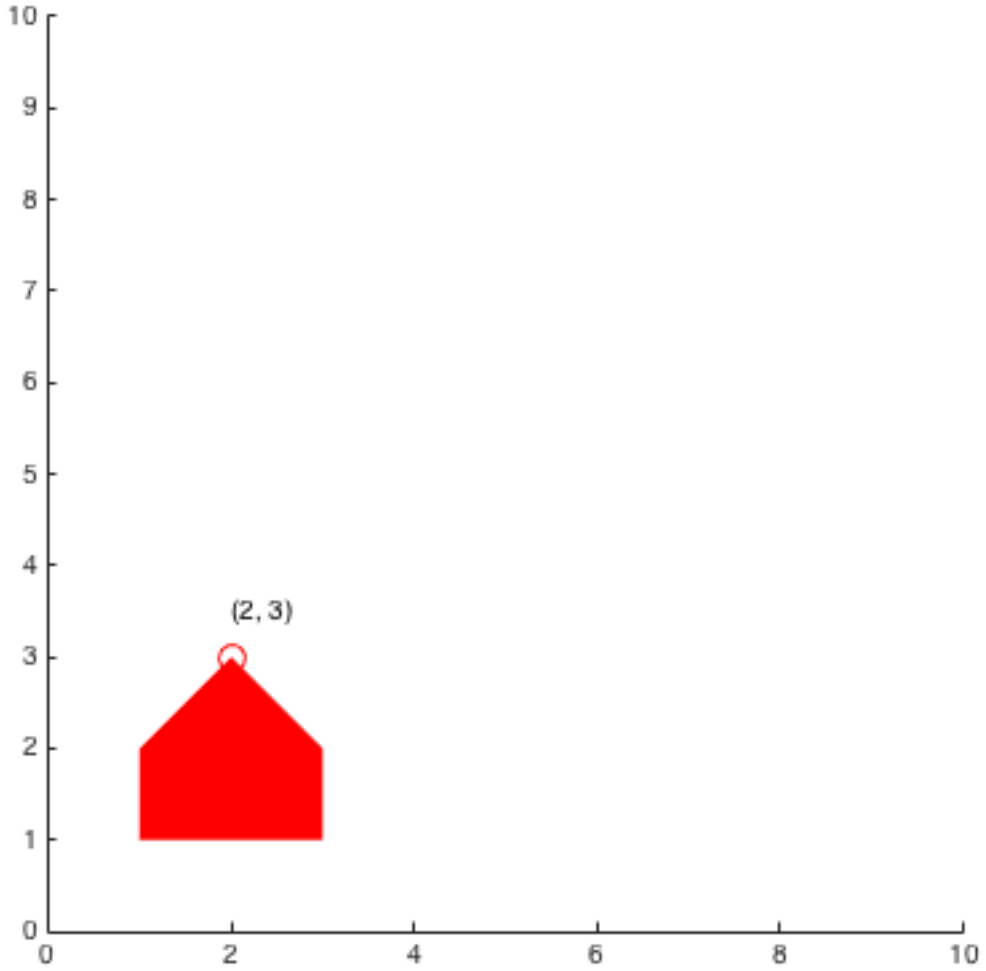
Penn

Intro

- **In the motion planning problems we have considered so far we have basically reduced the problem to planning on a graph where the robot can take on various discrete positions which we can enumerate and connect by edges.**
- **In practice most of the robots that we build can move continuously through space. Configuration space is a handy mathematical and conceptual tool which was developed to help us think about these kinds of problems in a unified framework.**

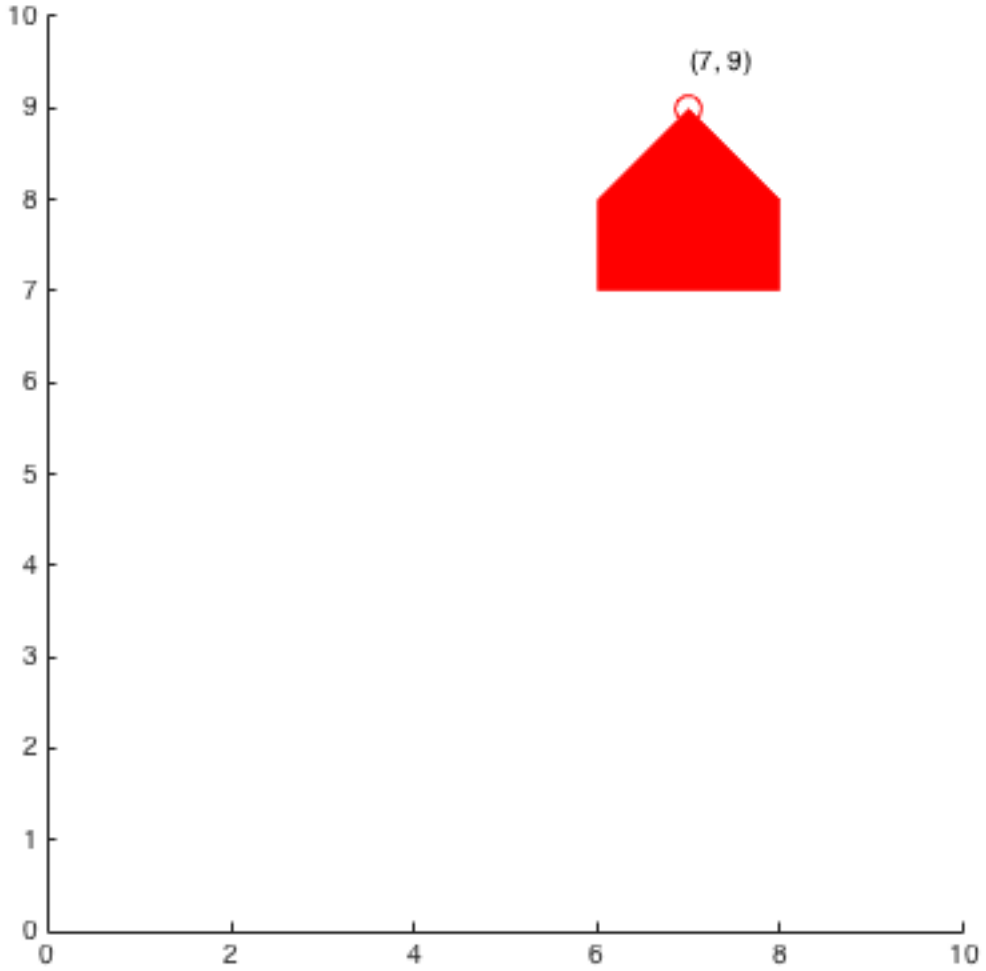
- **Basically the configuration space of a robot is the set of all configurations and or positions that the robot can attain.**
- **This slide shows a simple example of a robot that can translate freely in the plane. Here we can quantify the positions that the robot can take on with a tuple (tx, ty) which denotes the coordinates of a particular reference point on the robot with respect to a fixed coordinate frame of reference.**

Simple Robot – Translation in the Plane

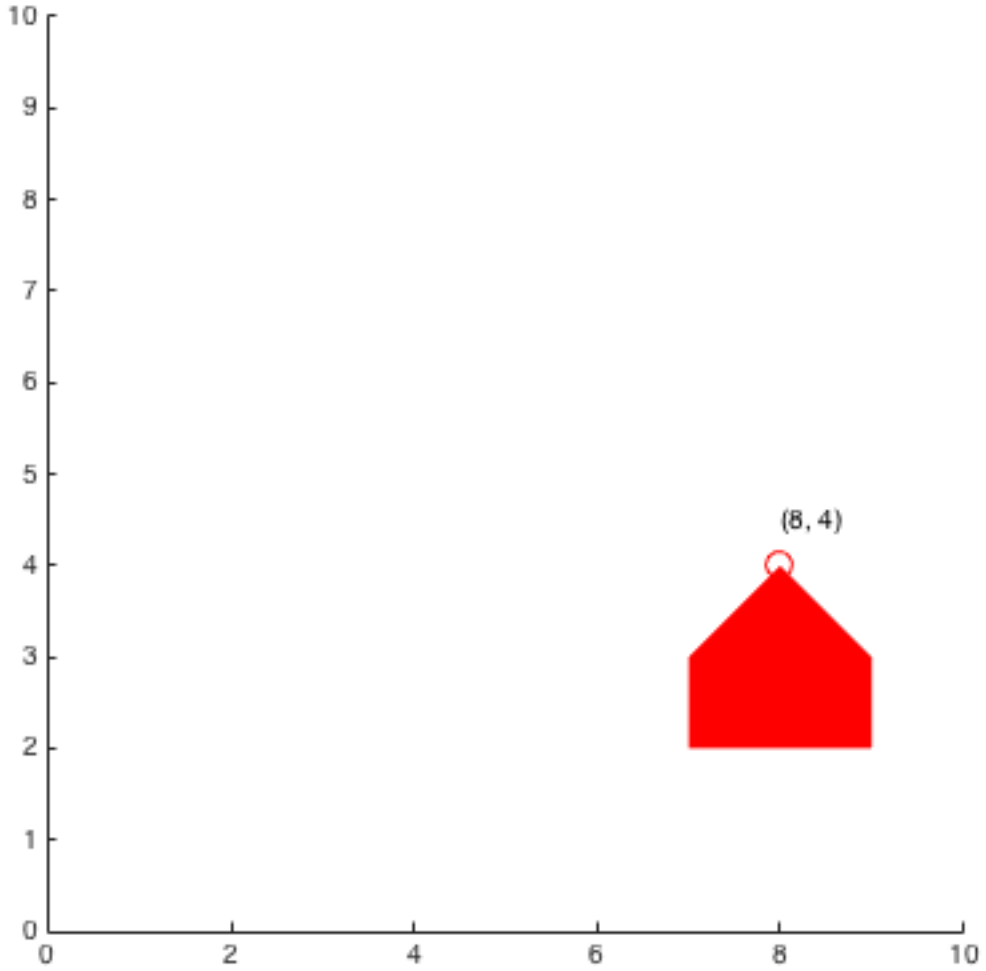


- **Here are a couple of configurations that this translating robot can take on along with the associated coordinates.**

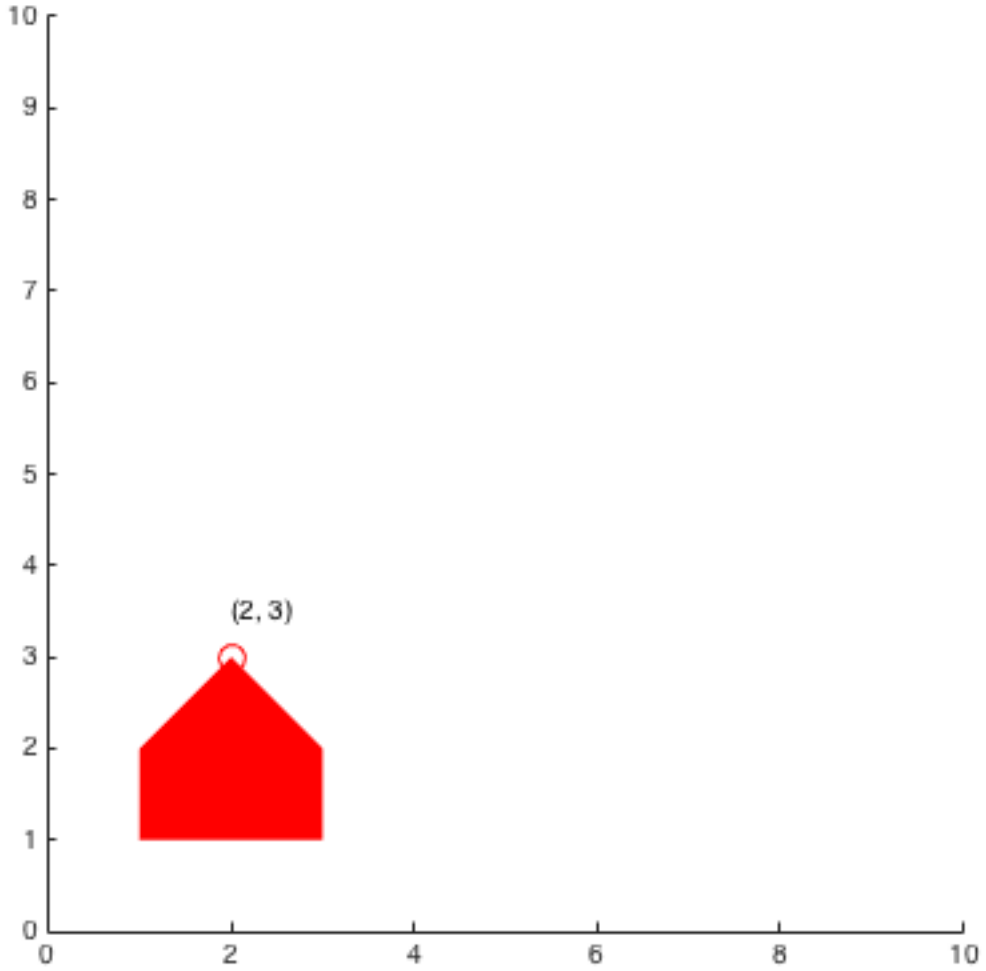
Simple Robot – Translation in the Plane



Simple Robot – Translation in the Plane

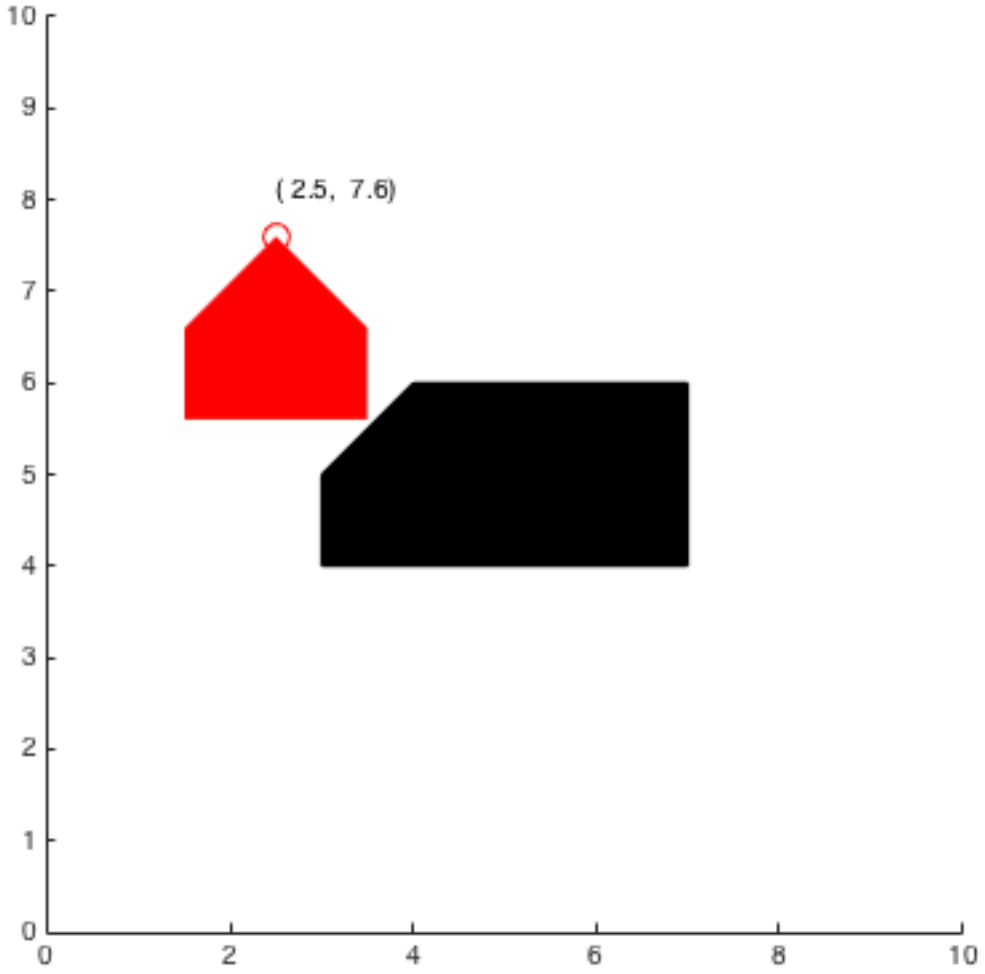


Simple Robot – Translation in the Plane



- **In this case we would say that our robot has two degrees of freedom and we can associate the configuration space of the robot with the points on the 2D plane – namely these (tx,ty) coordinates.**

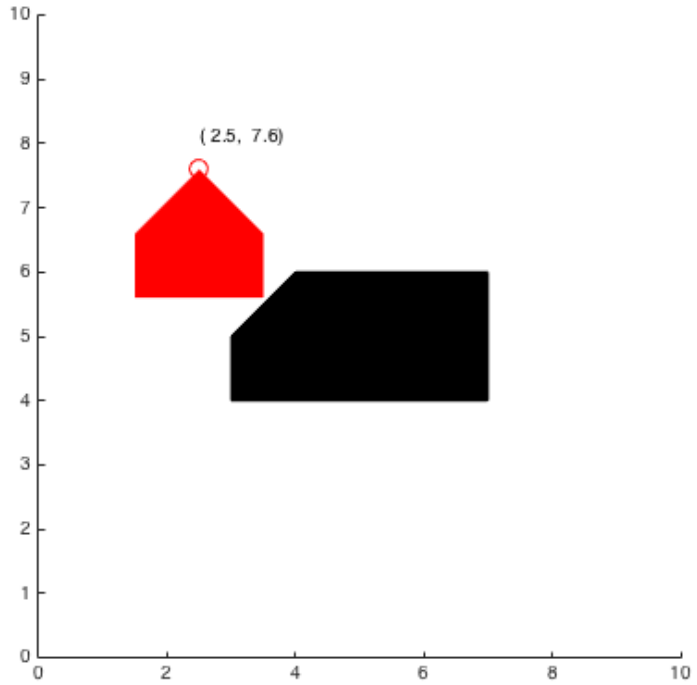
Adding an Obstacle

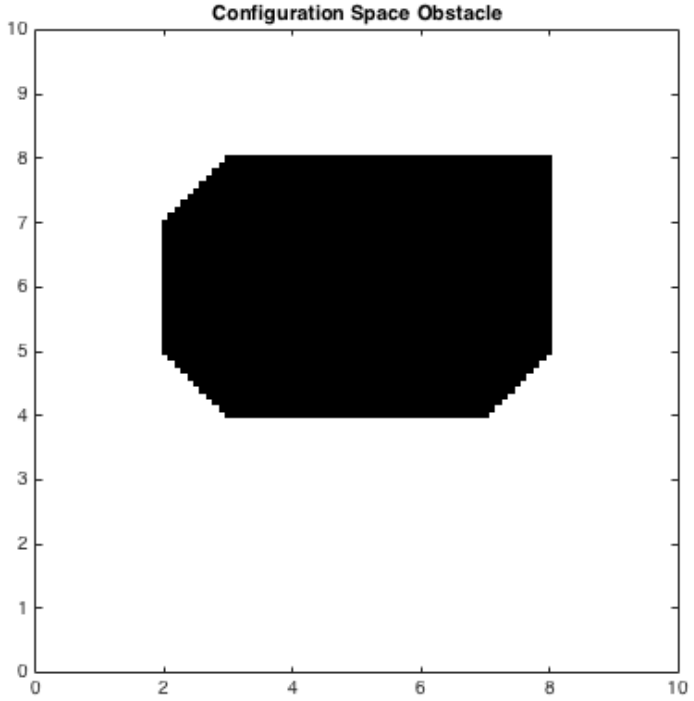


- Now we will make the story a little bit more interesting by introducing fixed obstacles into our model.
- What these obstacles do is make certain configurations in the configuration space unobtainable. This figure shows the (tx,ty) configurations that the robot CANNOT attain because of the obstacle
- This set of configurations that the robot cannot inhabit is referred to as a CONFIGURATION SPACE OBSTACLE.
- Conversely the region of configuration space that is outside of the configuration space obstacle is termed freespace.

- On the right hand side of this figure we plot the configuration space obstacle corresponding to the geometric obstacle shown in the left half.
- Again the C-space obstacle denotes the set of configurations that the robot CANNOT attain because of collision with the obstacle.

Configuration Space Obstacle

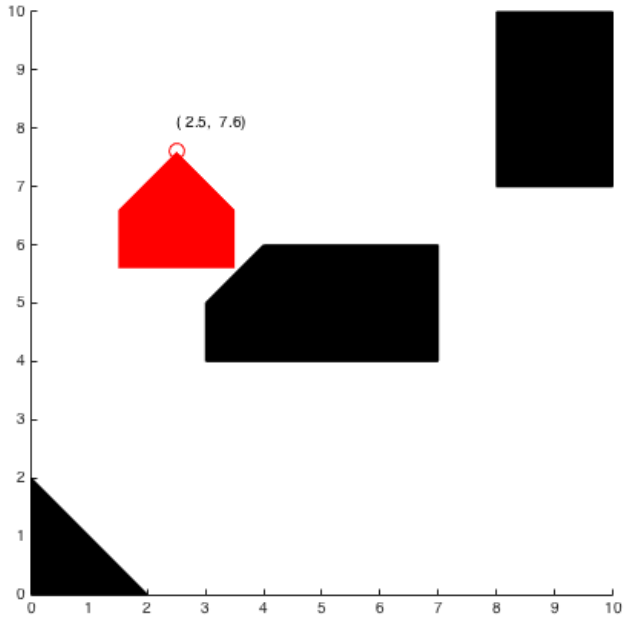


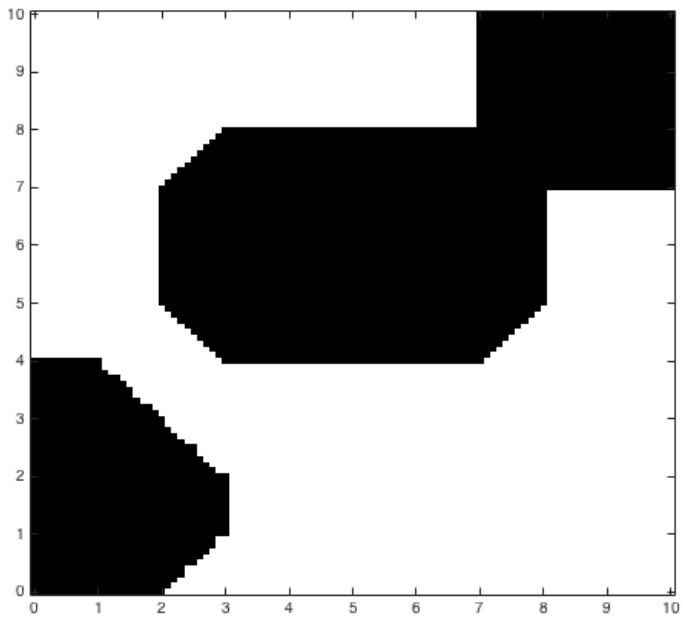


- **Note that the dimensions and shape of the configuration space obstacle are obtained by considering both the obstacle and the shape of the robot.**
- **More formally the configuration space obstacle in this case is the Minkowski sum of the obstacle**

- **If we have multiple obstacles in space we can visualize the union of all of the configuration space obstacles and we get a picture like this.**
- **Again the configuration of the robot corresponds to a point in the space and the dark areas correspond to configurations that the robot cannot attain.**

Configuration Space Obstacles





- **In this setting the task of planning a path for our robot correspond to planning a trajectory through configuration space from the starting configuration to the end configuration.**

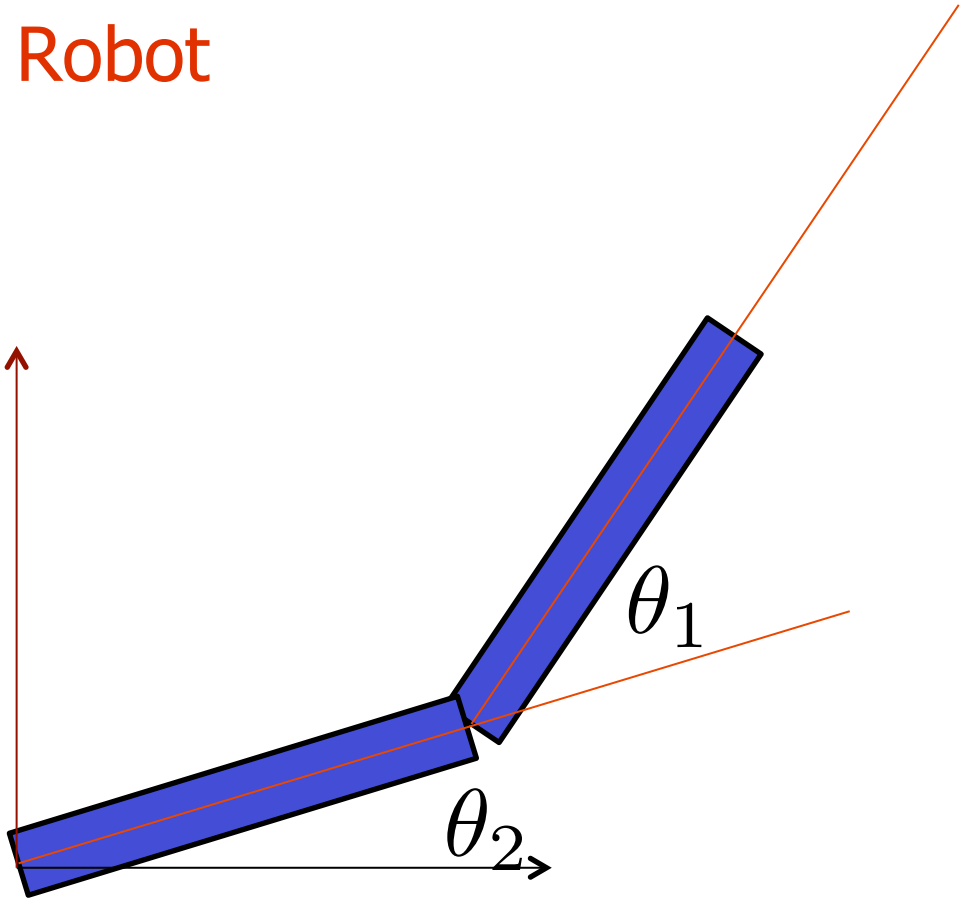
RR Arm

SECTION 2.2

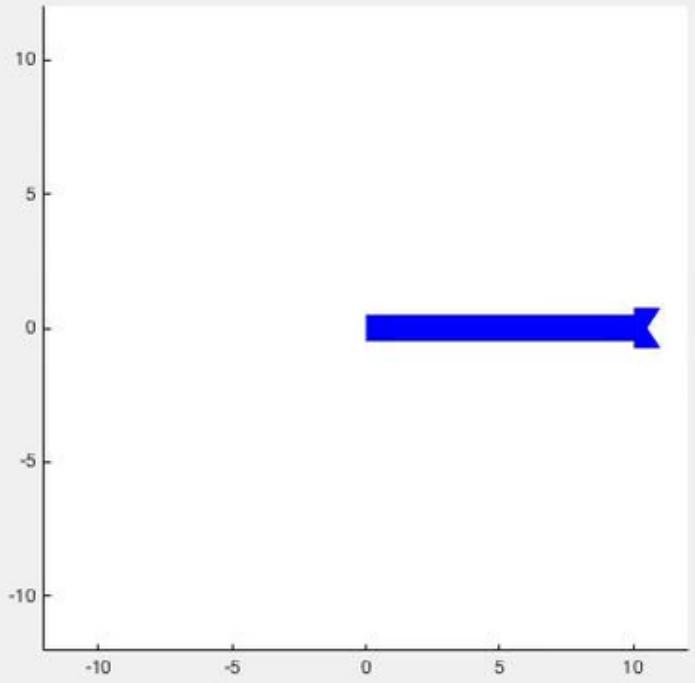
Penn

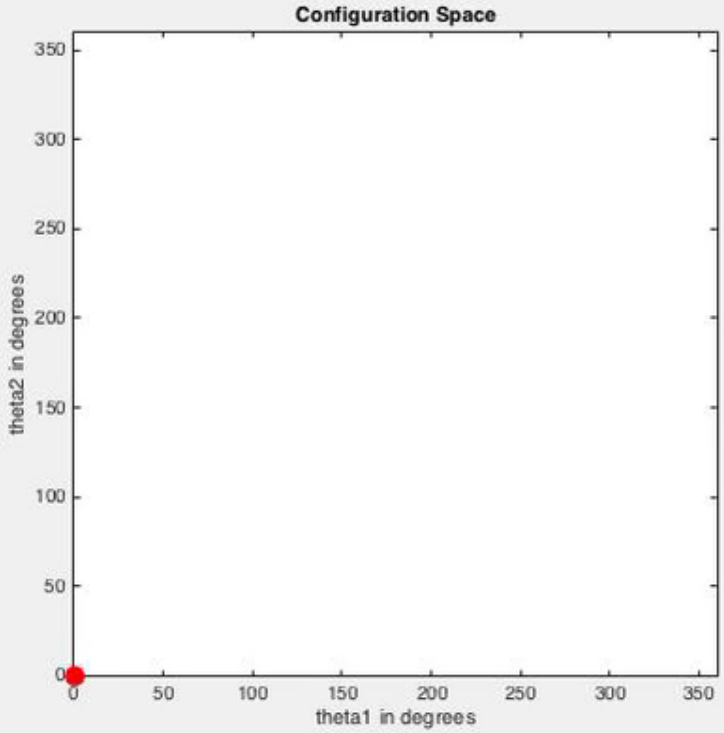
- Configuration space is a general concept can be applied to lots of robots
- This slide shows a simple planar arm with two revolute joints Marked 1 and 2.
- Here again we can think of all of the possible configurations of this robot and associate them with a tuple of joint angles (theta_1, theta_2).
- In this case the two angles can freely range from 0 to 360 degrees.
- This movie shows our planar 2 link robot moving around along with the corresponding trajectory in configuration space

Two Link Planar Robot



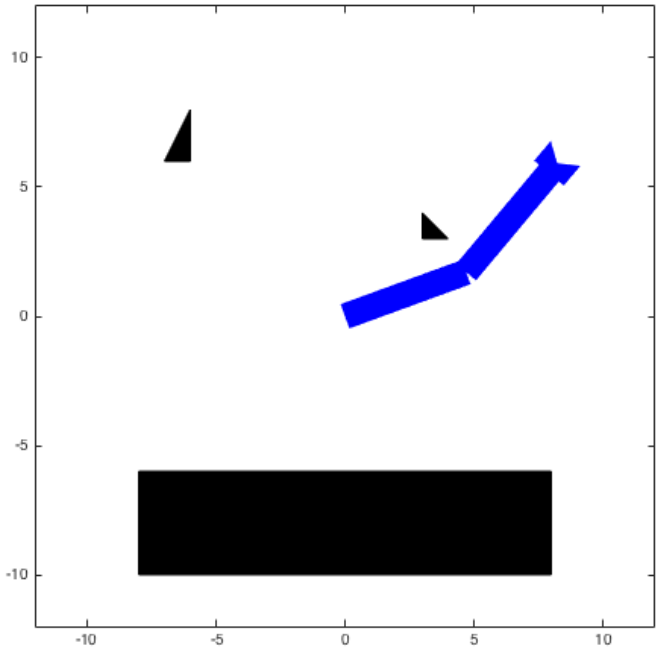
Two Link Robot + Configuration Space

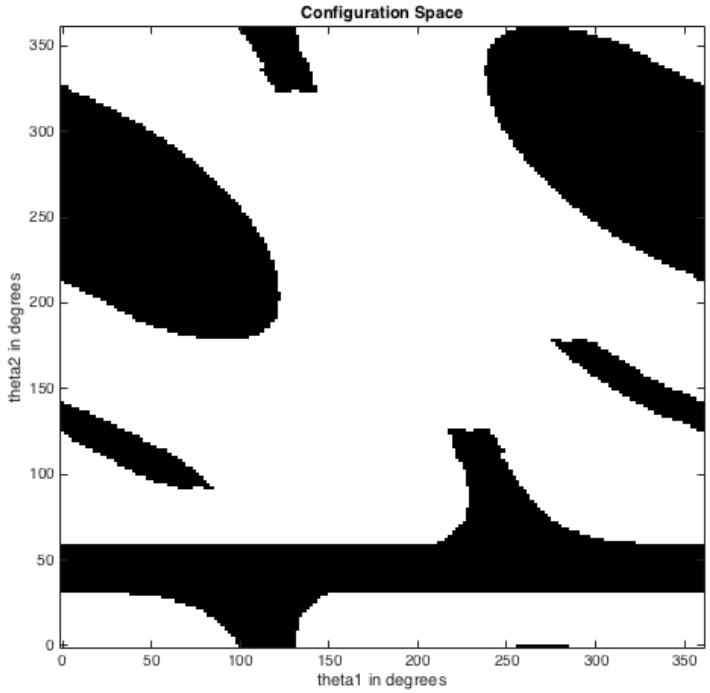




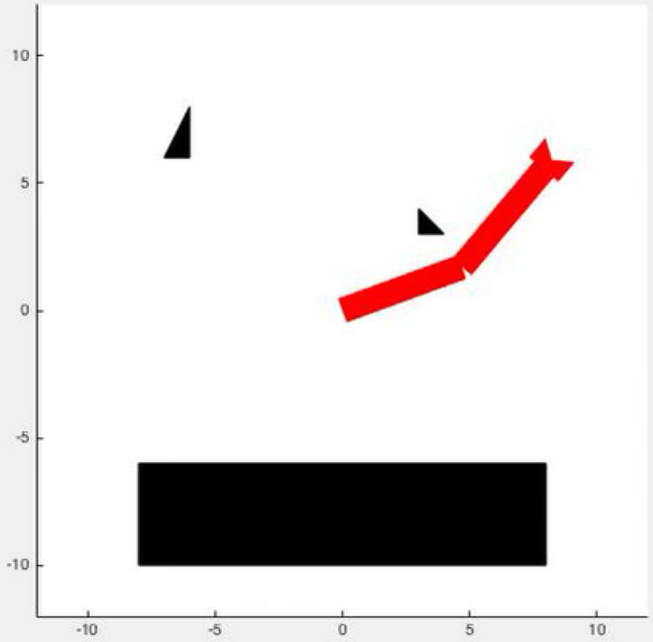
- Once again we can introduce obstacles into the environment and consider which configurations become infeasible because of collision.
- This figure shows the robot, the obstacles and the corresponding situation in configuration space
- Note that because of the way we have chosen coordinates for C-Space the simple polygonal obstacles actually turn into interesting looking c-space obstacles.

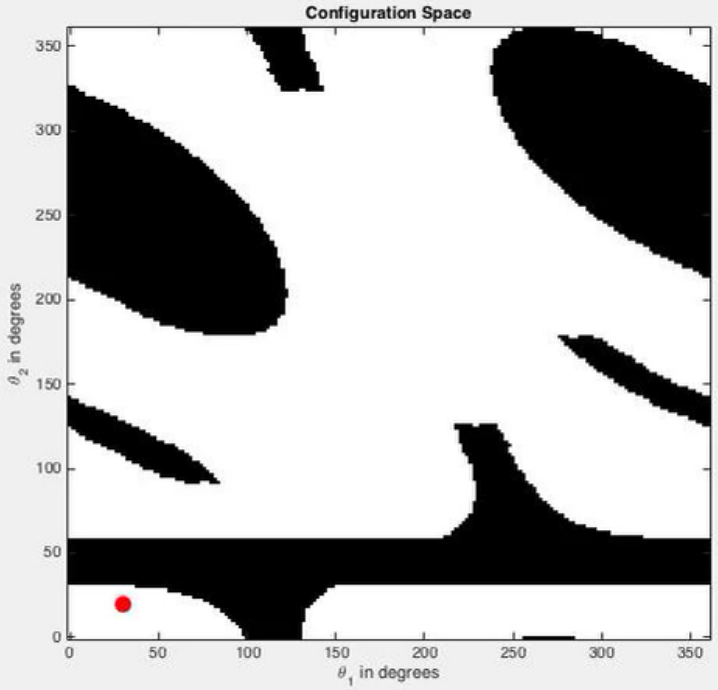
Configuration Space Obstacles for RR Arm





Planning Trajectory for 2 Link Arm

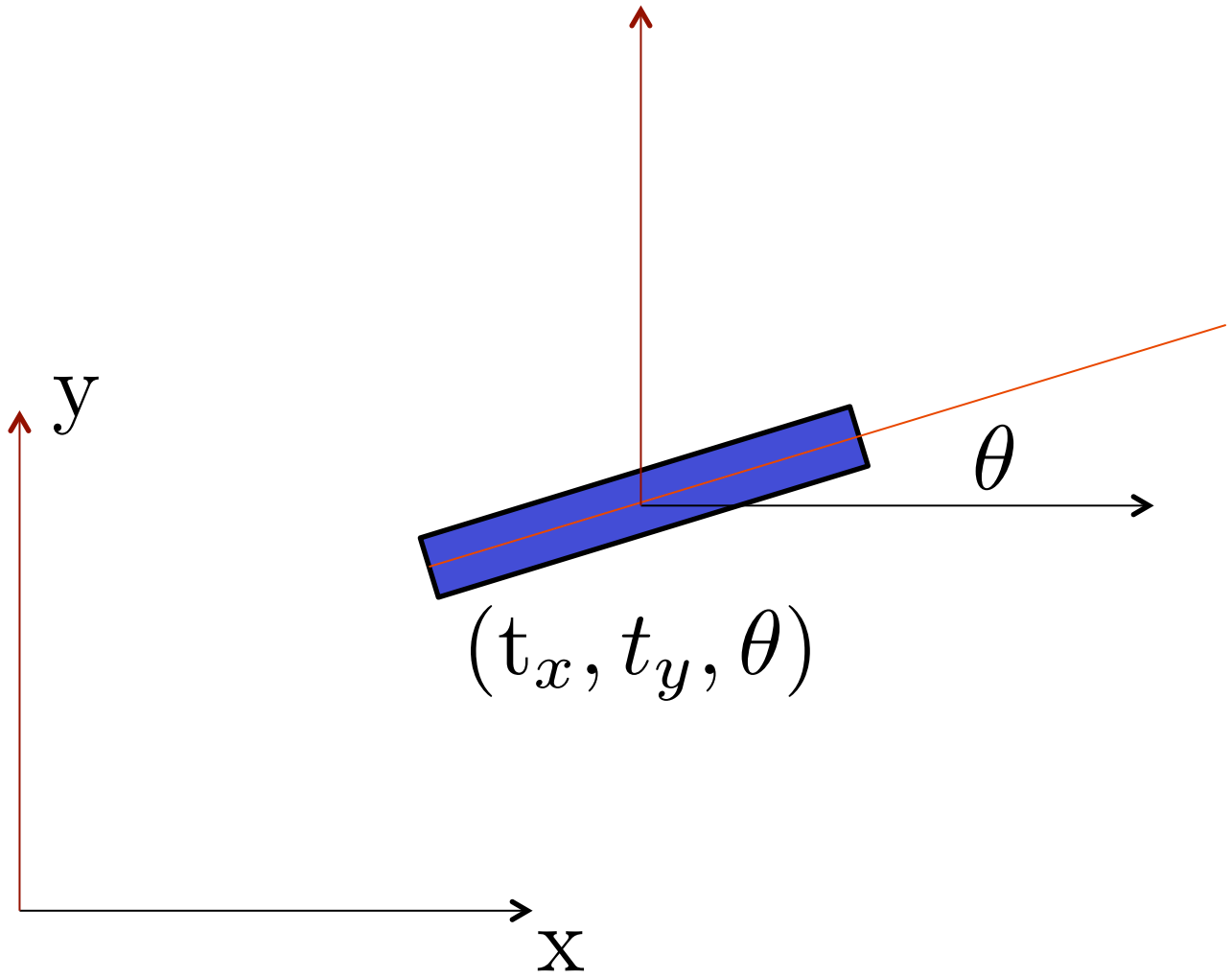




Piano Mover's problem

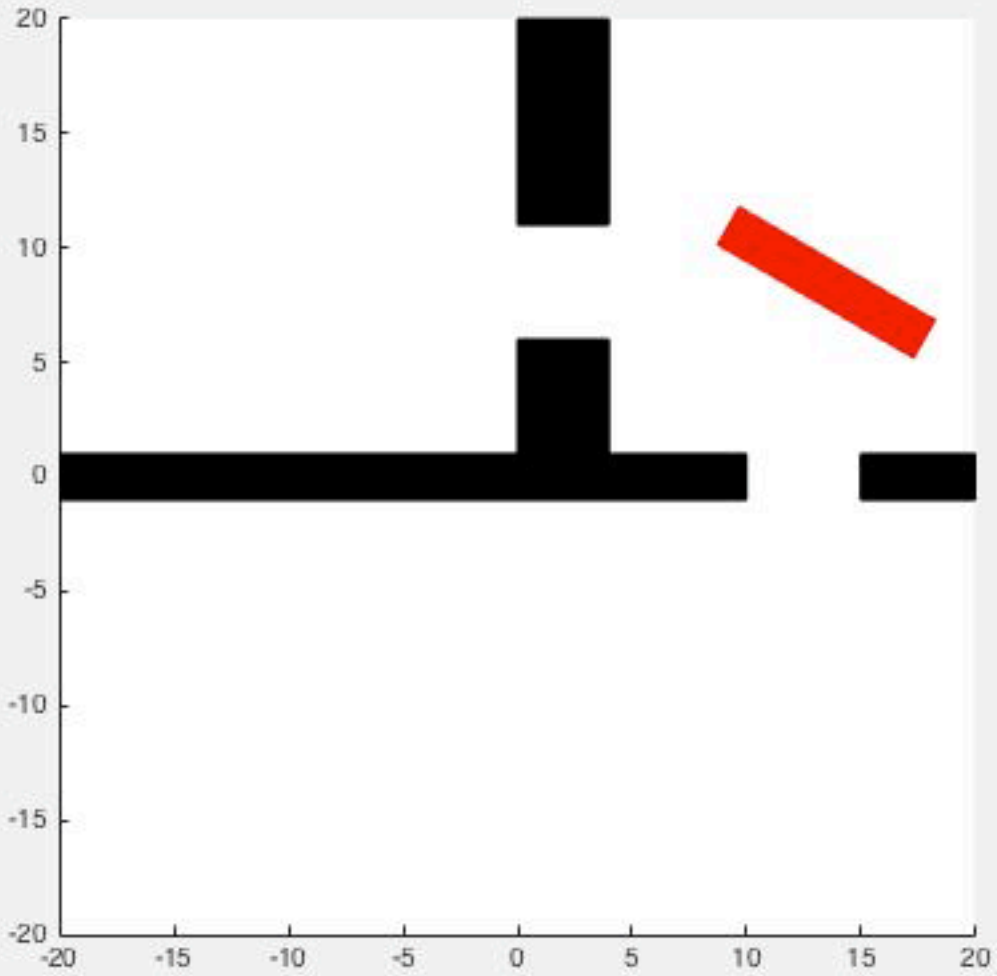
SECTION 2.3

Translating and Rotating Robot

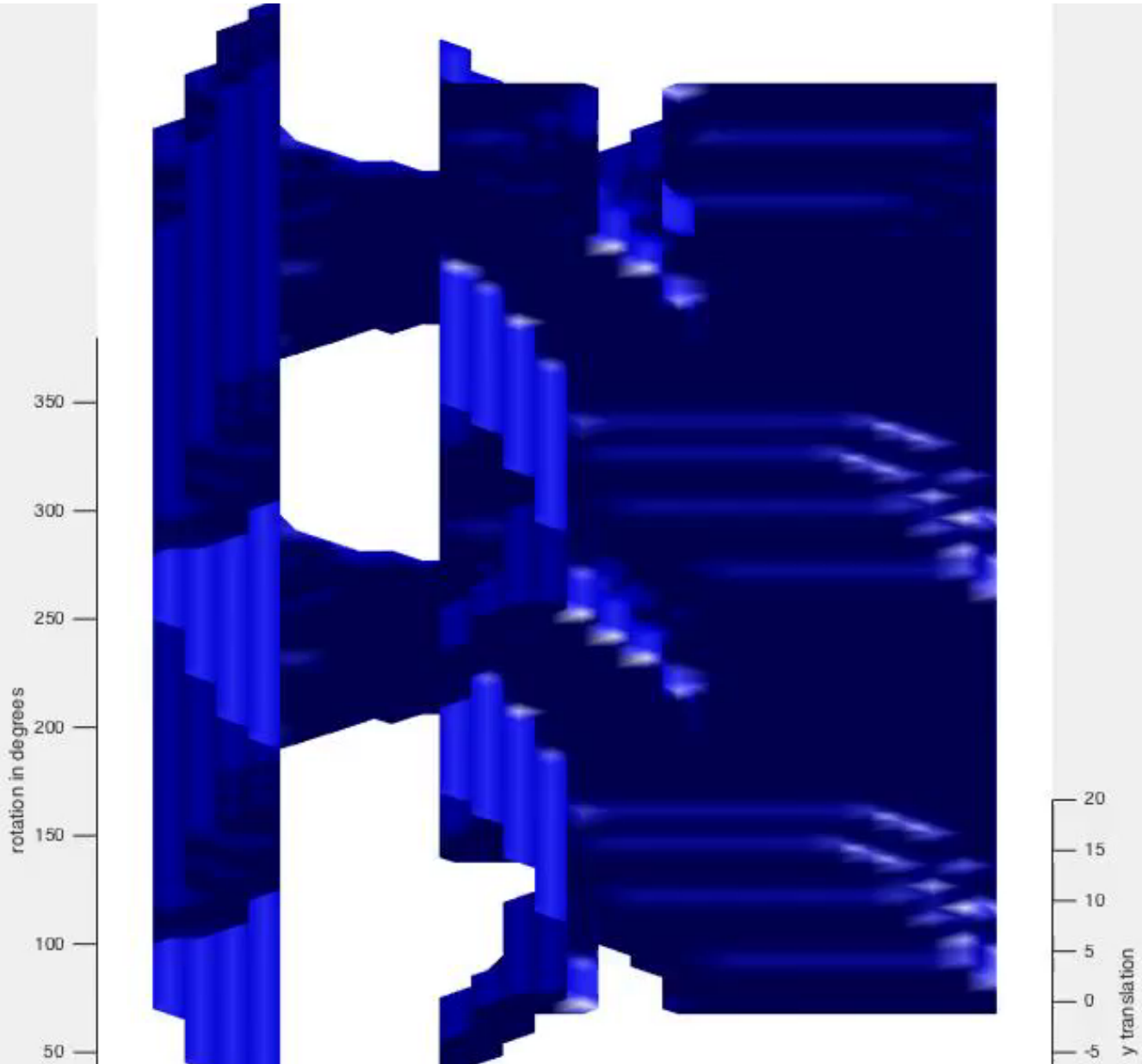


- **Once again when we introduce obstacles into the workspace we can think about the set of configurations that are eliminated. In this case the configuration space has 3 dimensions and the configuration space obstacles can be thought of as 3 dimensional regions in this space.**
- Except in this case the configuration space is 3D so visualization is a little more challenging.

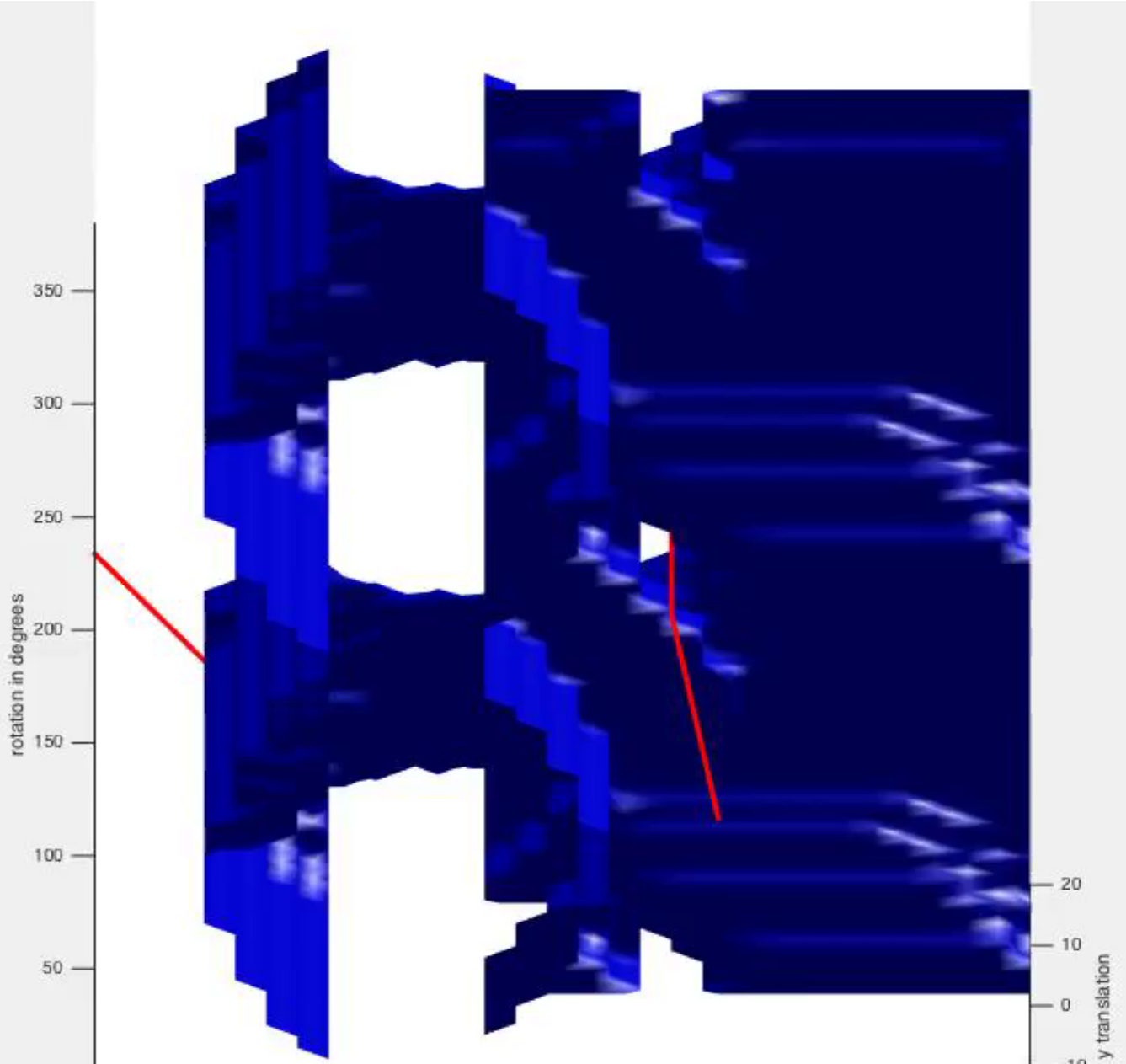
Rotation and Translation with Obstacles



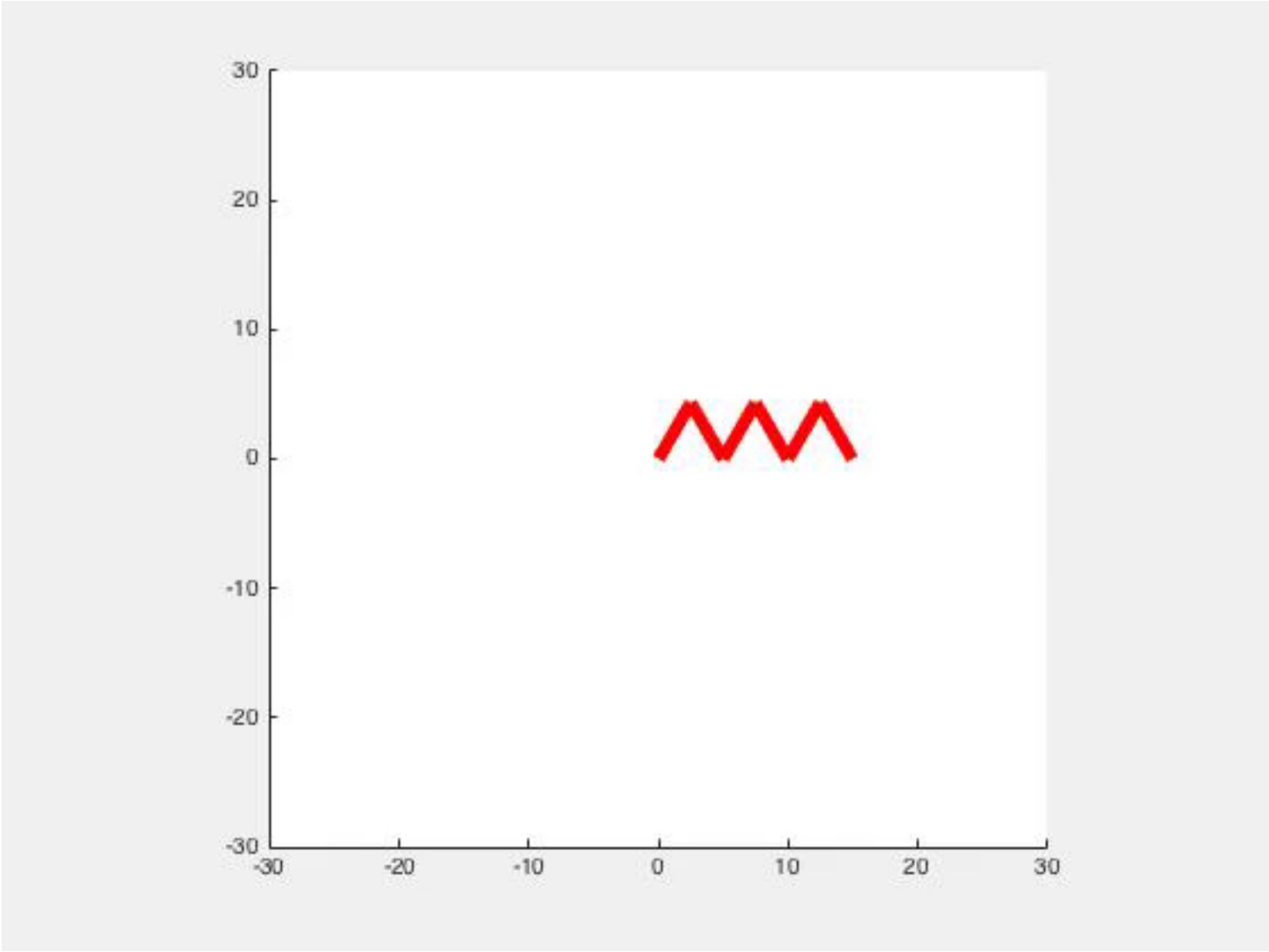
Configuration Space Obstacle



Configuration Space Trajectory



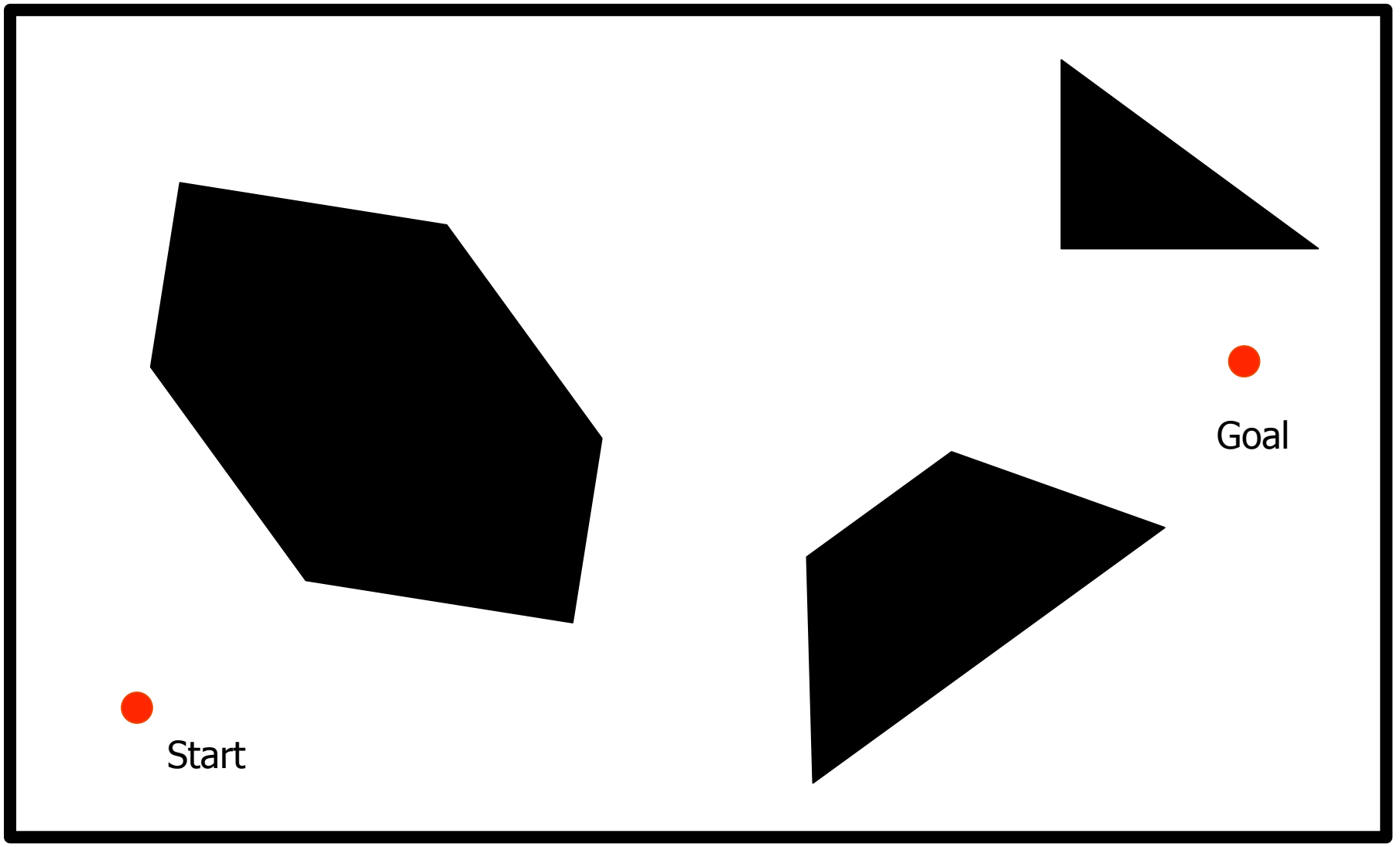
Six Link Planar Robot

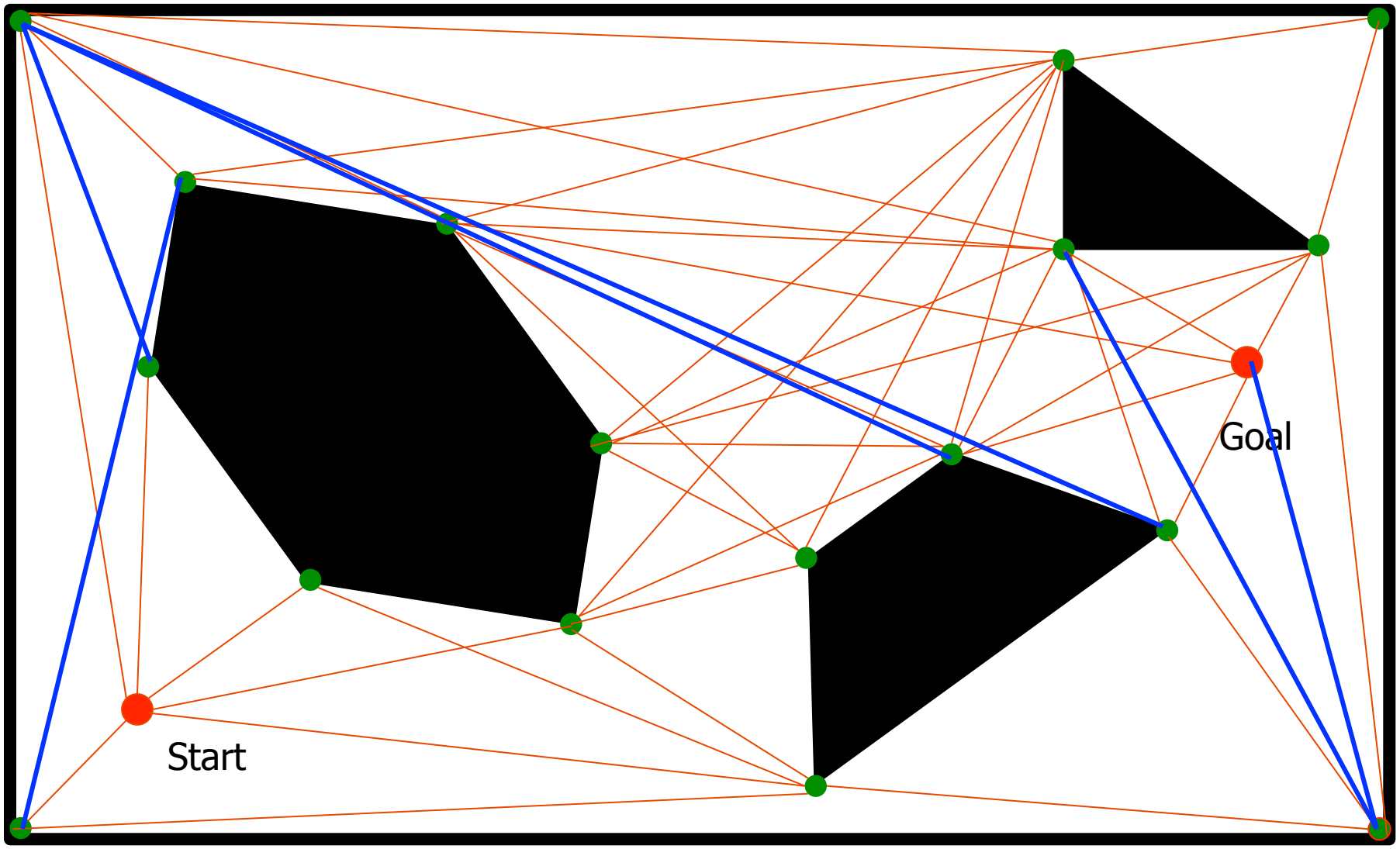


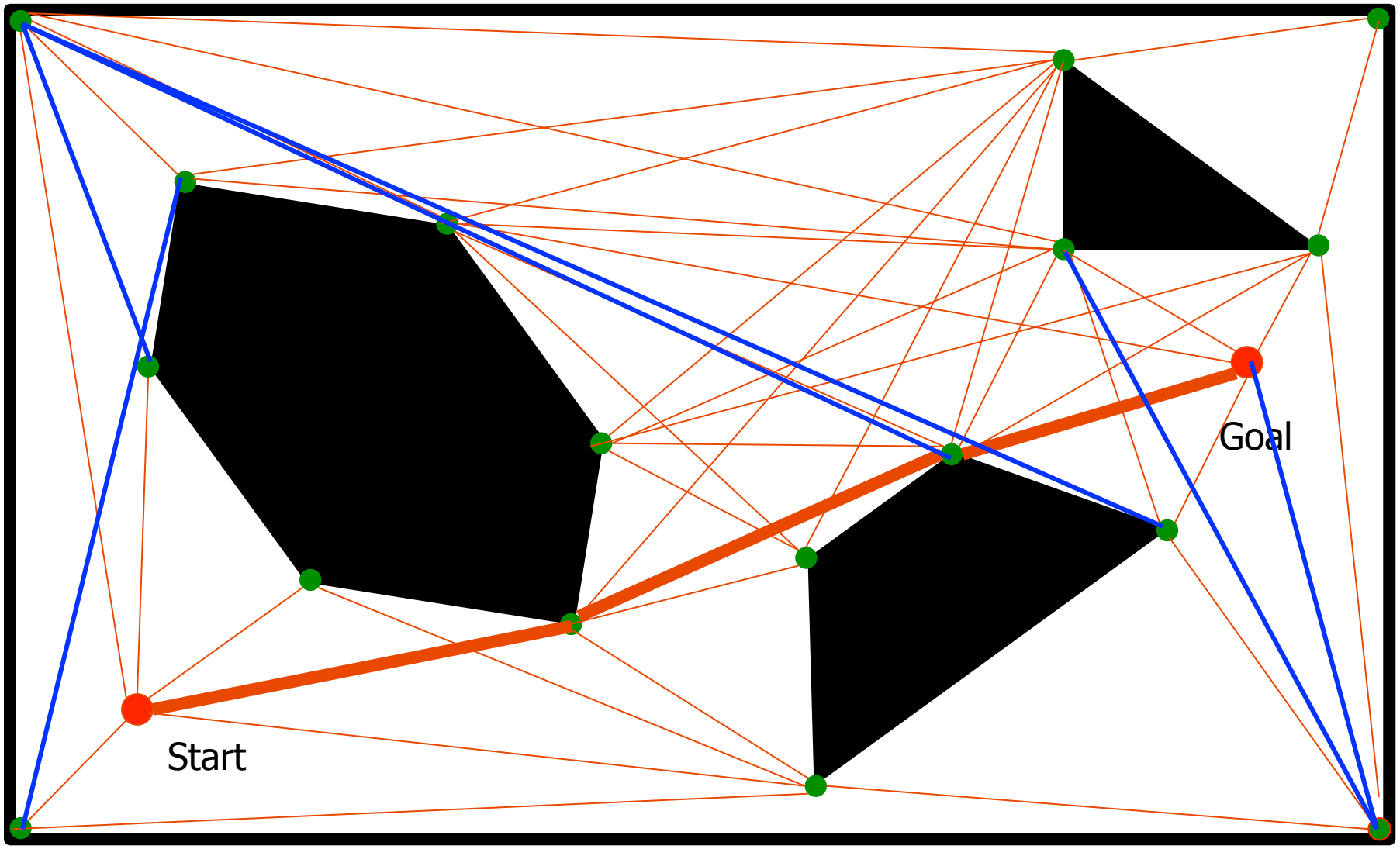
Visibility Graph

SECTION 2.4

Penn


Goal
Start


Goal
Start


Goal
Start

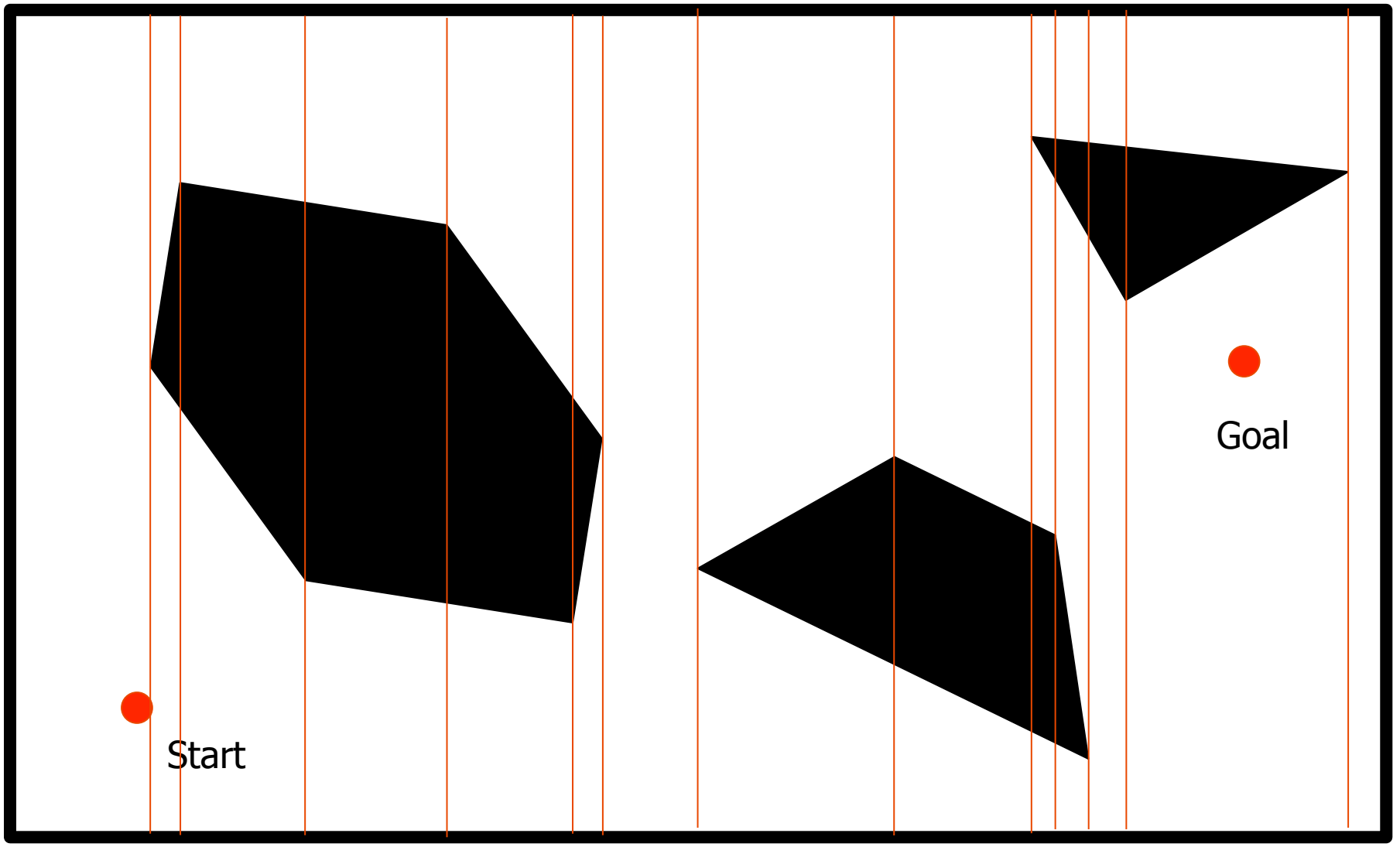
- **This visibility graph algorithm is actually complete. That is it will find a path if one exists and report failure if no path can be constructed. This could happen if the start or destination is surrounded by an obstacle.**

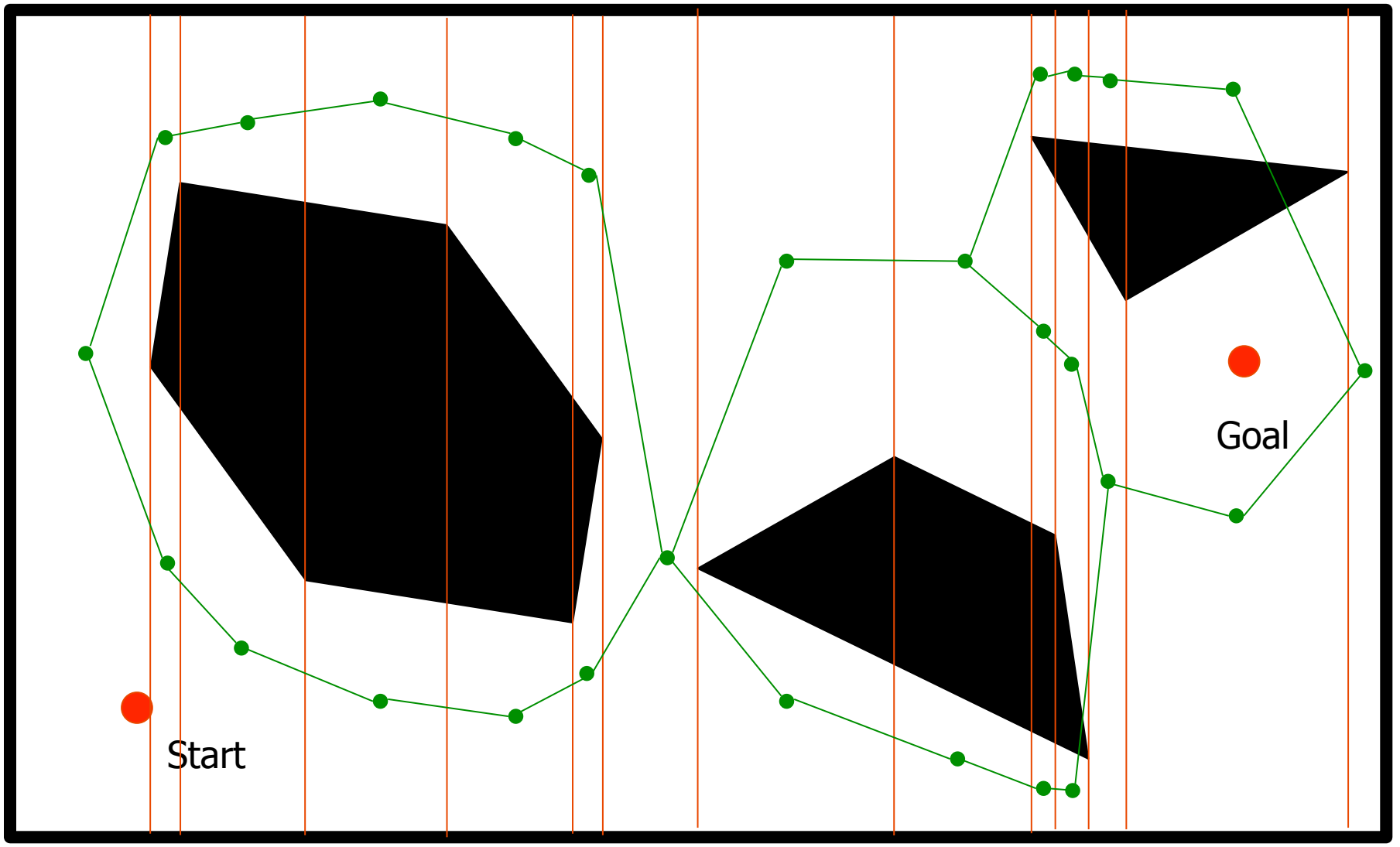
- **Moreover you can prove that this algorithm will actually construct the shortest possible path between the two points. You can see the intuition for this by thinking of the path between the two vertices as a piece of string, imagine what would happen if you pulled this string as tight as possible to eliminate all of the slack, the resulting trajectory would consist of a series of straight lines between vertices corresponding exactly to the edges of the visibility graph.**

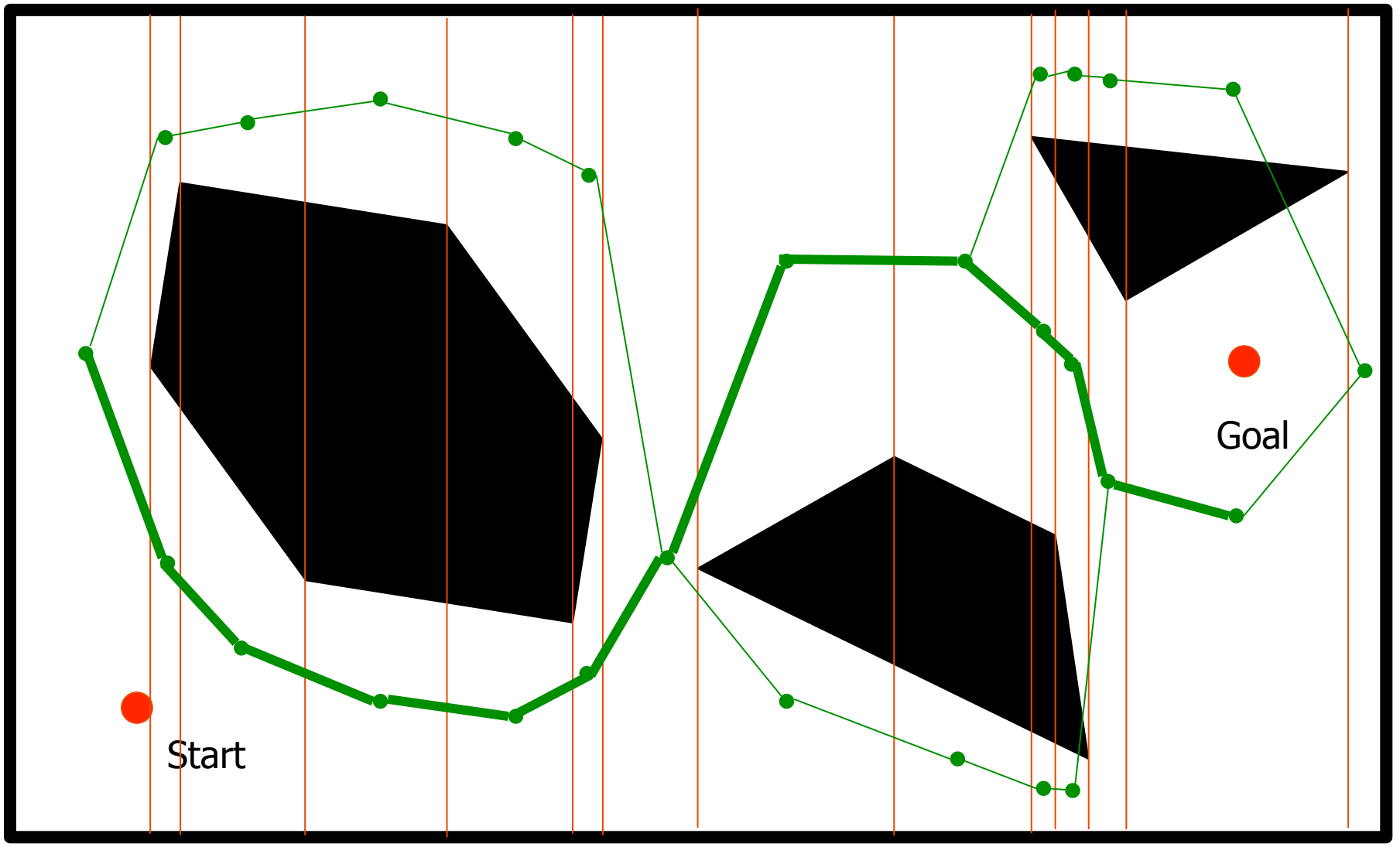
Trapezoidal Decomposition

SECTION 2.5

Penn


Goal
Start


Goal
Start


Goal
Start

Collision Detection and Planning

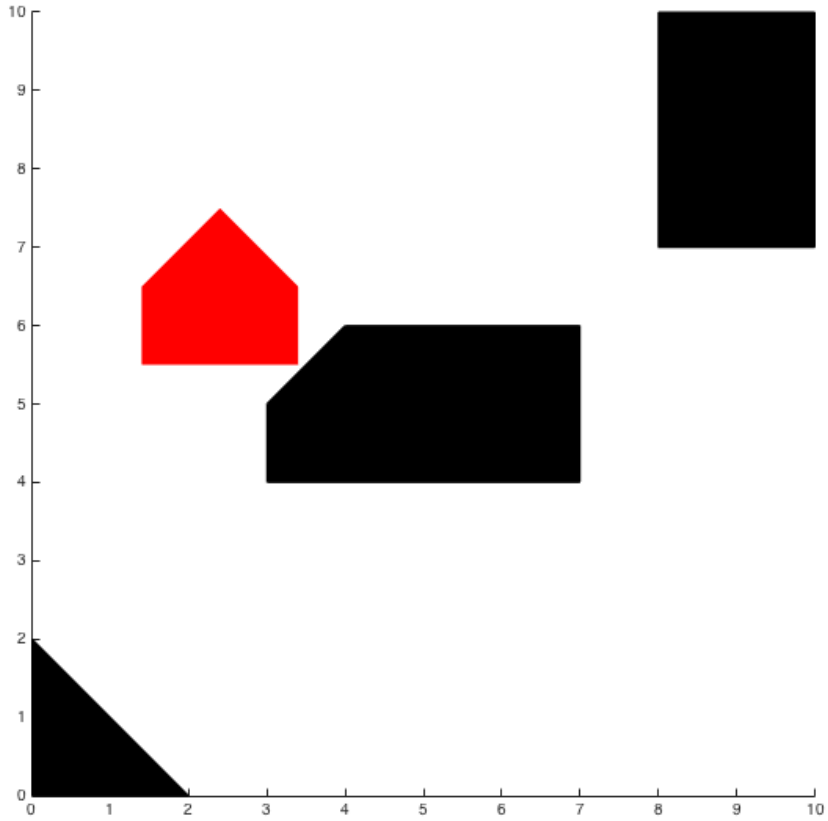
SECTION 2.6

- **Polygonal obstacles are convenient to work with because they provide an explicit description of the configuration space obstacles.**
- **Oftentimes we do not have this luxury and the obstacles are instead defined implicitly by a collision function.**

Collision Detection Function

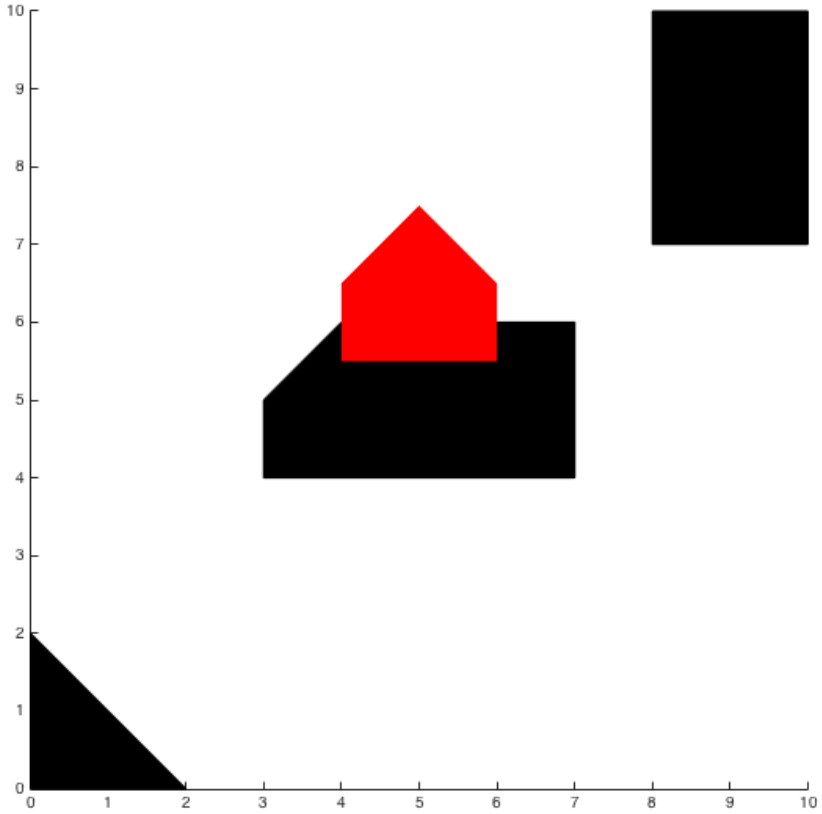
- Let x denote the coordinates of a point in configuration space.
- CollisionCheck (x) should return 0 if x is in freespace and 1 if x results in a collision with the obstacles.

Collision Detection



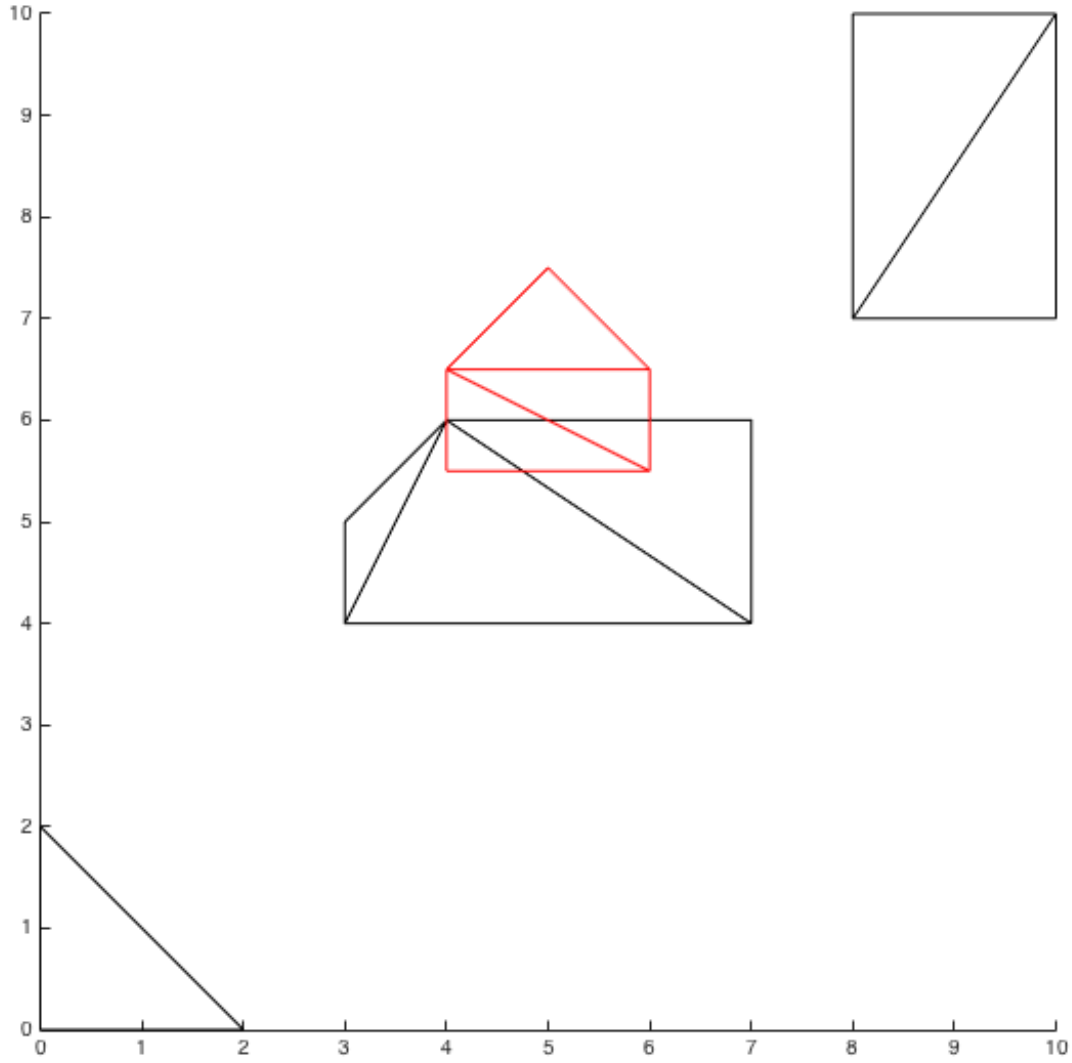
CollisionCheck (x) -> 0

Collision Detection



CollisionCheck (x) -> 1

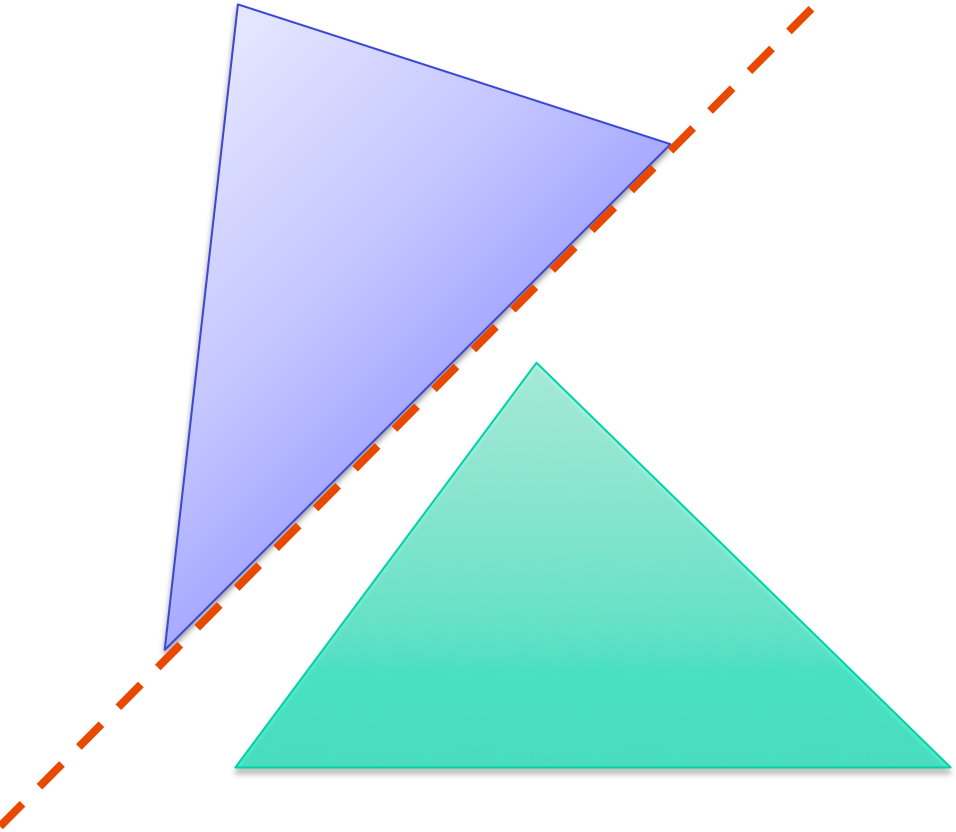
Collision Detection



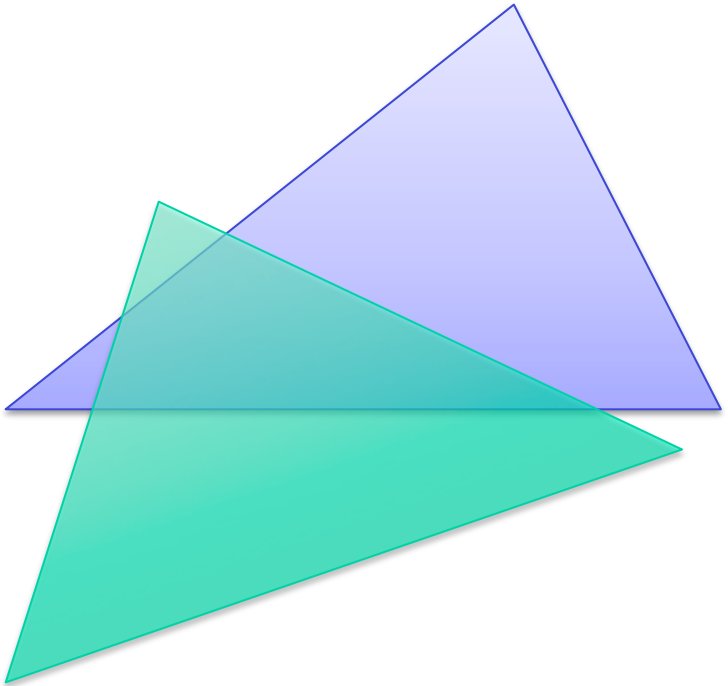
- **Deciding whether the robot and the obstacle intersect is now a matter of determining whether any of the robot triangles overlap any of the obstacle triangles.**

- **Here we can make use of the fact that triangles are convex polygons. In this circumstance it means that we can test whether two triangles intersect by checking all of the sides on both triangles and testing whether that side acts as a separating line where all of the points from one triangle lie on one side and all those from the other lie on the opposite side.**
- **If you can find such a separating edge you have proved that the triangles don't intersect, if not you can conclude that they do.**

Testing Triangles for Overlap



Separating Edge -> No Overlap

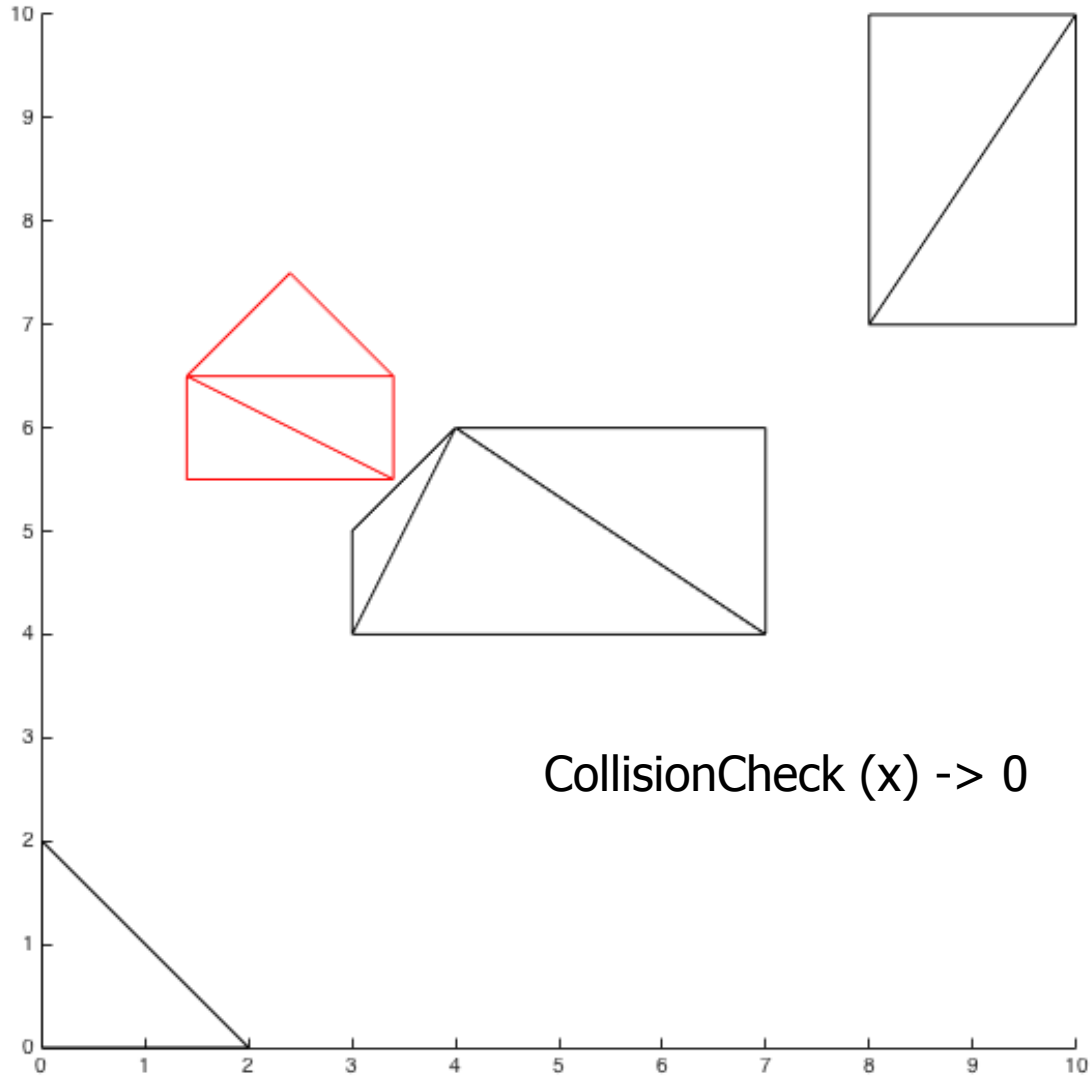


No Separating Edge -> Overlap

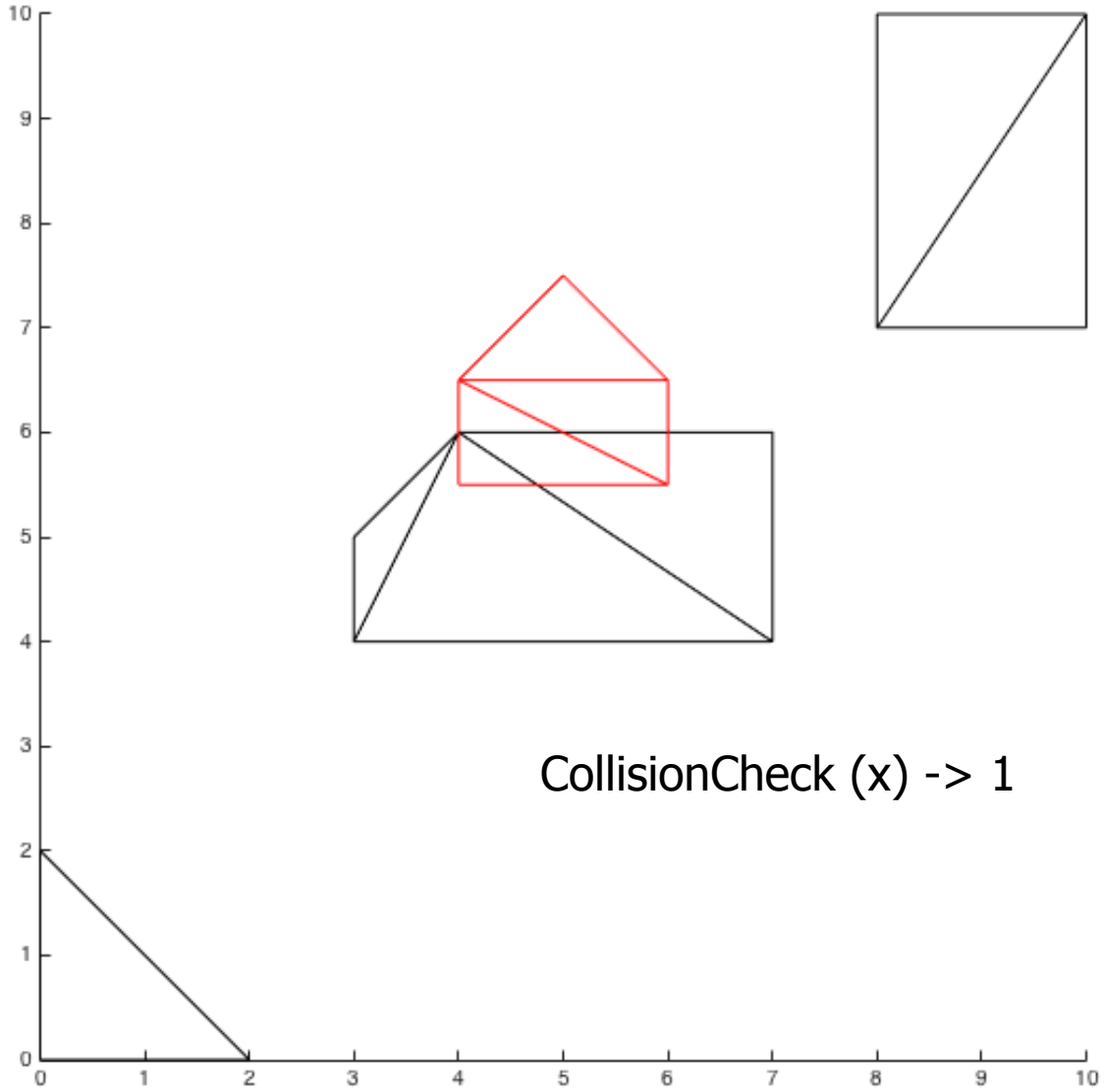
- **This idea of finding a separating line or plane actually generalizes to higher dimension. For instance if you have convex polygons in three dimensions like boxes or pyramids you can check for collision by testing if any of the faces act as separating planes.**

- **In the cases that we have considered so far the robots and the obstacles are basically composed of simple polygons, so to test for collision we first transform the robot according to the configuration space parameters and then test for collision between the robot components and the obstacles.**

Collision Checking



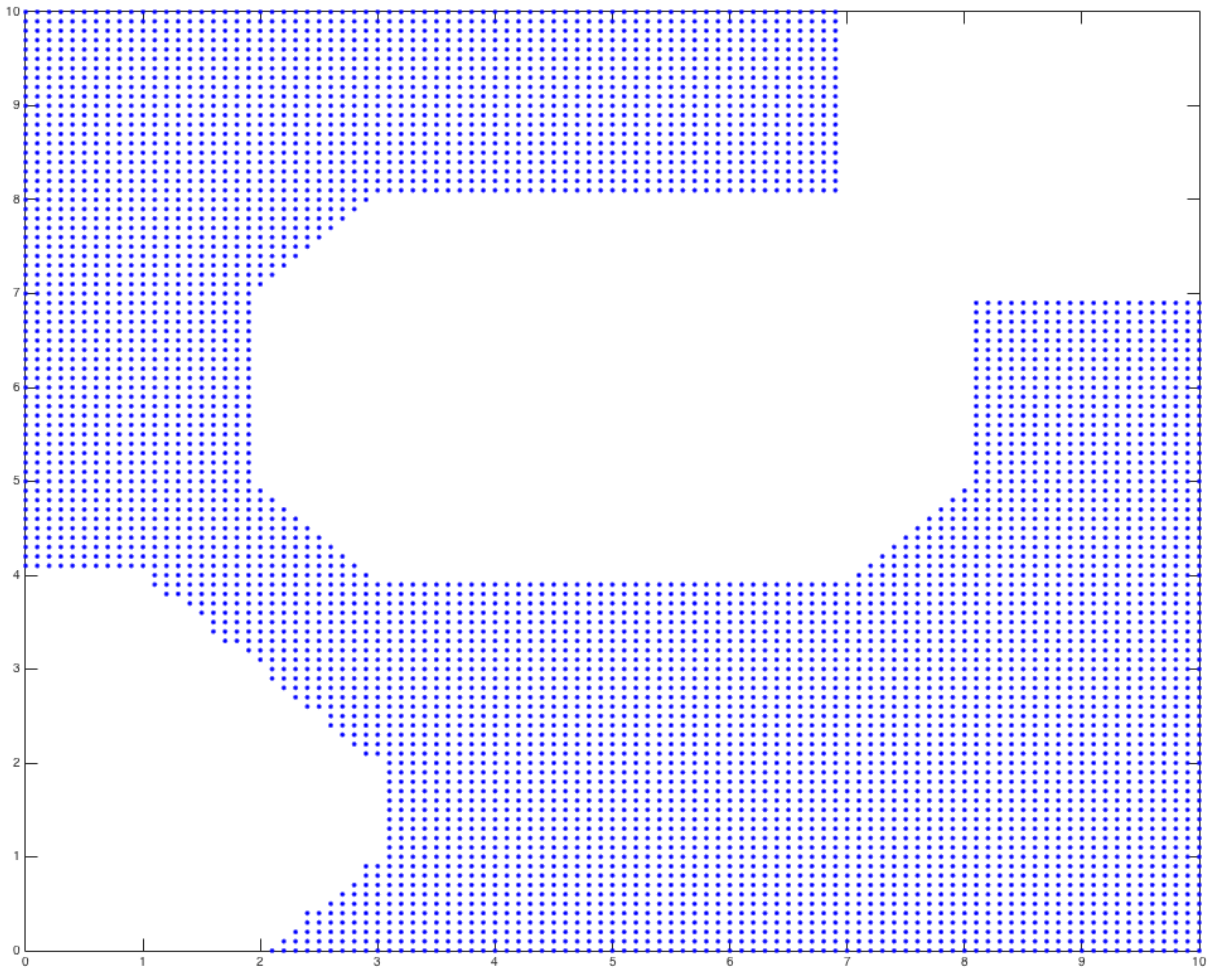
Collision Checking



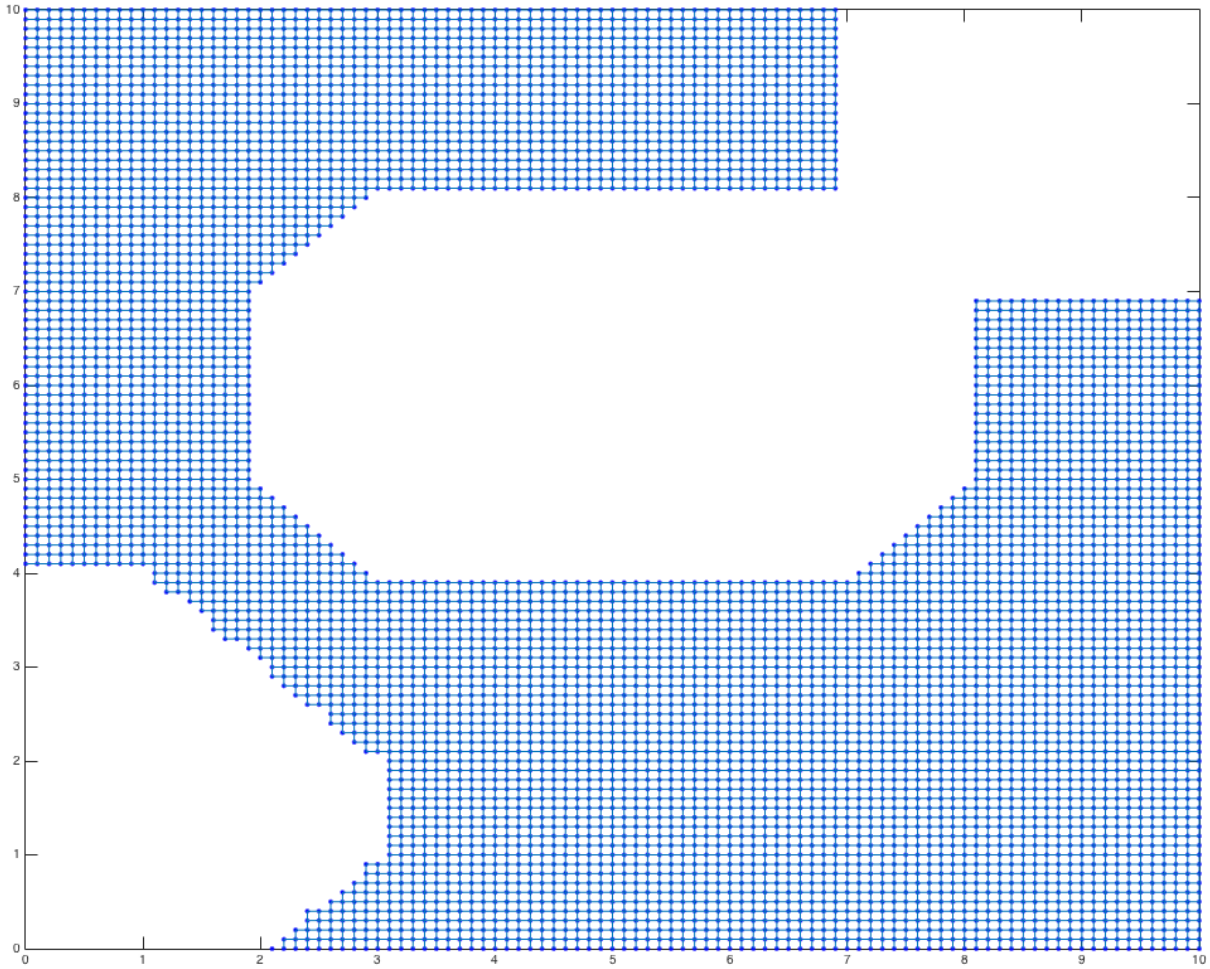
CollisionCheck (x) -> 1

Penn

Sampled Points in freespace



Graph Made from Sampled Points



Path through graph